Measure Energy Consumption using Python

Phase 3 submission Document

PROJECT TITLE: Measure energy consumption

Phase 3 : Development Part 1

INTRODUCTION

- Green computing technology focuses on the efficient use of computing resources. In computing devices such as laptops, smartphones, tablets, or other mobile devices, energy consumption is the top priority because they are run on battery, with limited lifespan, as their source of power (Banerjee et al. 2007).
- With the increasing complexity of IT equipment, the energy consumption rate of these devices system also increases (Silven and Jyrkka, 2007).
- Most portable mobile device users are conscious of the energy usage by these devices and consequently, they look for ways through which the lifespan of the battery can be extended to serve them longer (Rahmati et al. 2007).

- Experiments relating to energy measurement could be at various levels: the hardware level; energy efficiency directive level (Simunic, et al. 2000); operating system (Sagahyroon, 2006); software application or data and user levels (Ravi, et al. 2008). Energy conservation is made possible through the use of different techniques which estimate or forecast energy consumption at the device and application level (Krintz, et al. 2004).
- The goal of green computing technology is to reduce carbon emission, maximize performance and prolong the lifespan of the computing resources

1. Literature Review

- The Smart2020 report (The Climate Group and GeSI, 2008) predicts an increasing trend of BAU CO2 emissions for the ICT industry. The emissions growth rate for three ICT categories (end-user devices, telecommunication and networks, and data centers) is expected to decrease from 6.1% 3.8%. By 2020, the ICT industry's footprint is expected to rise to 1.3 GtCO2e (equivalent to 2.3% of global emissions by 2020).
- The PC (e.g. desktops, laptops, etc.) footprint (due to its embodied and usage emissions) is the highest (60%) followed by printers (18%), peripherals (13%), smartphones (10%), and tablets (1%). It is estimated that the footprint of end-user devices will grow at 2.3 percent per year to reach 0.67 GtCO2e in 2020 and thus, energy efficiency improvements in these devices and their proper usage are essential for reducing their overall footprint.

Energy Consumption of Media Players

- Modern technologies incorporate a number of power management features to reduce power waste. Dynamic Voltage and Frequency Scaling (DVFS) can enable the CPU speed to be dynamically varied based on the workload which leads to a reduced power consumption during periods of low utilization (Liu, et al., 2008).
- The energy-aware dynamic voltage scaling technique has been used to reduce energy consumption in portable media players (Yang & Song, 2009). This scheme showed a relationship between frame size and decoding time. These two cited work merely discuss how energy consumption can be reduced using various techniques, but have not measured the actual amount of energy being consumed by the

application. However, the energy consumption of Windows Media Player has been measured using the EEcoMark v2 tool (EecoMark, 2011) but the empirical details of the measurement have not been explicitly discussed. Media playback application power consumption has been analysed by Sabharwal (2011) using windows event tracing.

- Event tracing does not seem to be an appropriate method for measuring energy consumption because the process itself may have impact on the results. A comparative analysis of energy consumption of media players has been conducted by Techradar (2010).
- The energy consumption is monitored by playing a DVD on Windows Media Player (WMP) and VLC Media Player.
- Their research results show that the VLC Media Player is more energy efficient than Windows Media Player. However, the cited work has not mentioned which tool has been used for measurement and additionally, the experiment procedures have not been explitcitly discussed.

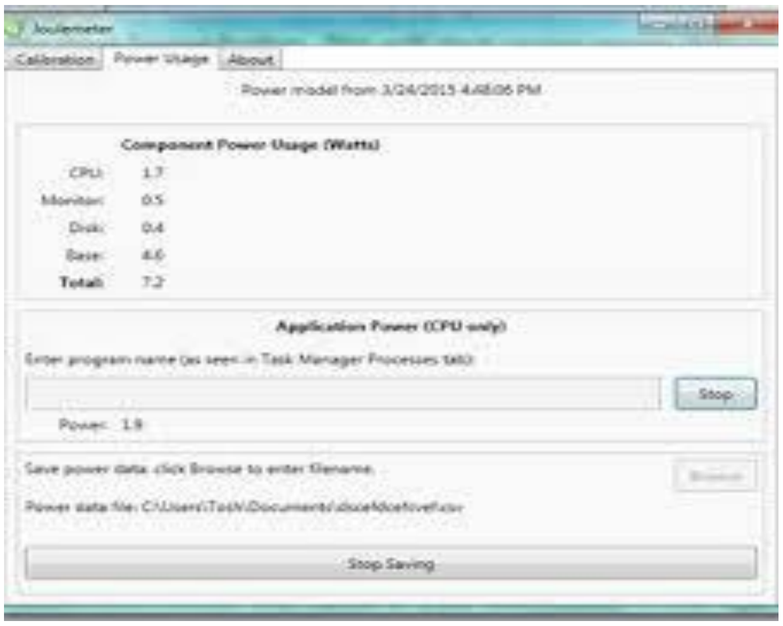
Tools and measure

- Power models are used to calculate the energy consumption of hardware and software. Kansal and Zhao (2008) use a generic automated tool to profile the energy usage of various resources components used by an application.
- This method is either too generic or coarse-grained and it is platform dependent (Seo, et al., 2007). The model proposed by Lewis and colleagues (2012) is an integrated model for the calculation of a system's energy consumption.
- More promising approaches are software energy measurement using energy application profiler (Noureddine, et al., 2013). In their contribution, Varrol and Heiser (2010) use Openmoko Neo Freerunner to decompose the energy consumption of each resource of a system. PowerAPI is an Application Programming Interface (API) used for monitoring the real time energy consumption of applications at the granularity of system process (Bourdon, et al., 2013).
- PowerAPI can also be used to estimate the energy consumption of a running process for hardware resources e.g. CPU or for hard disk or for both and many more other resources (Noureddine, et al., 2013). Energy consumption estimation in PowerAPI distinguishes the energy consumption for hardware resources and software blocks of codes.

Experimental Procedures

Experiment Set 1: To investigate the energy consumption of several web browser applications in Windows (the sample interface is shown in Figure 1) Experimental Steps

- Create a csv file for saving the real time power consumption data via Joulemeter (by clicking on the browse button);
- Click on the start saving button;
- Click on the start button to run the application in Google Chrome 1.3.27 (i.e a youtube video3);
- Click on the stop saving button to end the application; v. Repeat the above steps for 9 times;
- Repeat all the above steps for each of the following web browser: Internet Explorer 9;



Power model from 3/24/2015 4:48:06 PM

Component Power Usage (Watts)

CPU: 1.7
Monitor: 0.5
Disk: 0.4
Base: 4.6
Total: 7.2

Application Power (CPU only)

Enter program name (as seen in Task Manager Processes tab):

Power: 1.9

Save power data: click Browse to enter filename.

Power data file: C:\Users\Tosh\Documents\[illegible].csv

LOADING DATASET

Data loading in earthquake prediction is a crucial step in the process. It involves gathering and organizing relevant information to train and test predictive models

DOWNLOADING DATASET:

https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption

IMPORTING LIVRARIES

```
# Creating the model using full data and forecasting n steps aheaddeep_learner =
DeepModelTS(
data=d,
Y_var='DAYTON_MW',
lag=48,
LSTM_layer_depth=64,
epochs=10,
train_test_split=0
)# Fitting the model
deep_learner.LSTModel()# Forecasting n steps ahead
n_ahead = 168yhat = deep_learner.predict_n_ahead(n_ahead)
yhat = [y[0][0] for y in yhat]
```

Loading a dataset

- This section will discuss the results of the data analysis for the three sets of experiments discussed above. The Joulemeter monitored raw data is for the time stamp (in ms), power consumption (in Watts) for each component:

- CPU, monitor, disk, base and the application. The formula used to calculate the energy consumption by each component is: Energy (J) = Power (W) x Time (s). The results of the calculation for all the experiments runs are shown in Table 2. Note that the data is cleansed so as to omit records with application power consumption = 0W. If the number of remaining records > 50% of the raw data then the cleansed readings of the csv file will be included in the data analysis. However, if it is otherwise, then the experiment is considered an error (see Table 2). 4.1 Web Browser Applications for Windows Note: * - proportion of the remaining cleansed data > 50% of the raw data and ** is for otherwise and thus considered an error. White coloured records – results of experiments for Day 1; Blue coloured records – results of experiments for Day 2. Table 2: The hardware and application energy consumption for running several web browser applications Table 3 depicts the aggregated data for all the experiments conducted for each web browser: Google Chrome, Internet Explorer, Mozilla Firefox, and Safari. However, in order to provide a fair comparison among the web browsers, the time for running the application will have to be set to 1s (i.e. t = 1s) Consequently, the corresponding energy consumption for each component will have to be normalised for t = 1s (see Table 4). Based on the results shown in Table 4, it seems that Internet Explorer 9 consumes the least energy on laptops, followed by Mozilla Firefox and Safari while Google Chrome seems to be the highest energy consumer.

- These results are consistent with experiments conducted by the Center for Sustainable Energy Systems at Fraunhofer USA 4 , which compare the energy consumption of Internet Explorer 10, Mozilla Firefox and Google Chrome on laptops and desktops. Their results reveal that Google Chrome consumes the highest amount of energy followed by Mozilla Firefox. The conclusion drawn by them is that Internet

Program

```
import numpy as np

import pandas as pd




# Deep learning:

from keras.models import Sequential

from keras.layers import LSTM, Dense






class DeepModelTS():
```

```
"""
A class to create a deep time series model
"""
def __init__(
    self,
    data: pd.DataFrame,
    Y_var: str,
    lag: int,
    LSTM_layer_depth: int,
    epochs=10,
    batch_size=256,
    train_test_split=0
):


    self.data = data
    self.Y_var = Y_var
    self.lag = lag
    self.LSTM_layer_depth = LSTM_layer_depth
    self.batch_size = batch_size
    self.epochs = epochs
    self.train_test_split = train_test_split


@staticmethod
def create_X_Y(ts: list, lag: int) -> tuple:
    """
```

```
        A method to create X and Y matrix from a time series list for the
training of

        deep learning models

        """

        X, Y = [], []



        if len(ts) - lag <= 0:

            X.append(ts)

        else:

            for i in range(len(ts) - lag):

                Y.append(ts[i + lag])

                X.append(ts[i:(i + lag)])



        X, Y = np.array(X), np.array(Y)



        # Reshaping the X array to an LSTM input shape

        X = np.reshape(X, (X.shape[0], X.shape[1], 1))



        return X, Y



    def create_data_for_NN(

        self,

        use_last_n=None

        ):
```

```
"""
A method to create data for the neural network model
"""
# Extracting the main variable we want to model/forecast
y = self.data[self.Y_var].tolist()


# Subseting the time series if needed
if use_last_n is not None:
    y = y[-use_last_n:]


# The X matrix will hold the lags of Y
X, Y = self.create_X_Y(y, self.lag)


# Creating training and test sets
X_train = X
X_test = []


Y_train = Y
Y_test = []


if self.train_test_split > 0:
    index = round(len(X) * self.train_test_split)
    X_train = X[:(len(X) - index)]
```

```
            X_test = X[-index:]


            Y_train = Y[:(len(X) - index)]

            Y_test = Y[-index:]



        return X_train, X_test, Y_train, Y_test



    def LSTModel(self):
        """

        A method to fit the LSTM model

        """

        # Getting the data

        X_train, X_test, Y_train, Y_test = self.create_data_for_NN()



        # Defining the model

        model = Sequential()

        model.add(LSTM(self.LSTM_layer_depth, activation='relu',
input_shape=(self.lag, 1)))

        model.add(Dense(1))

        model.compile(optimizer='adam', loss='mse')



        # Defining the model parameter dict

        keras_dict = {

            'x': X_train,

            'y': Y_train,
            'batch_size': self.batch_size,

            'epochs': self.epochs,

            'shuffle': False

        }



        if self.train_test_split > 0:

            keras_dict.update({

                'validation_data': (X_test, Y_test)

            })



        # Fitting the model

        model.fit(

            **keras_dict

        )



        # Saving the model to the class

        self.model = model



        return model



    def predict(self) -> list:

        """

        A method to predict using the test data used in creating the class
        """
        yhat = []

        if(self.train_test_split > 0):

            # Getting the last n time series
            _, X_test, _, _ = self.create_data_for_NN()

            # Making the prediction list
            yhat = [y[0] for y in self.model.predict(X_test)]

        return yhat

    def predict_n_ahead(self, n_ahead: int):
        """
        A method to predict n time steps ahead
        """
        X, _, _, _ = self.create_data_for_NN(use_last_n=self.lag)

        # Making the prediction list
        yhat = []
```

```
for _ in range(n_ahead):

    # Making the prediction

    fc = self.model.predict(X)

    yhat.append(fc)



    # Creating a new input matrix for forecasting

    X = np.append(X, fc)



    # Ommiting the first variable

    X = np.delete(X, 0)



    # Reshaping for the next iteration

    X = np.reshape(X, (1, len(X), 1))



 return
```

- To obtain a benchmark for comparison against the performance of the recurrent network models, two models were developed. The first is the last observation carry forward (LOCF).
- It is a naïve model in which the one-step ahead prediction is the last observed value in the sequence. The MAE calculated for the LOCF model on the validation split using a simple 2:1 split is 3.983.
- Despite being extremely simple, the LOCF model otherwise also known as simple persistence forecasting, is nonetheless a credible benchmark as research in forecasting has shown. Outperforming the LOCF model

when the variation in the data is high is in reality difficult. The second benchmarking base model is support vector regression. Since the method of support vector machines(SVR) was proposed in 1995 [33], SVR is a popular machine learning model proposed for forecasting tasks even recently

Conclusion

In this integrated research, we have demonstrated the different tools that could be used to measure the power and battery consumption of various web browser and stand alone applications. The Joulemeter has been employed for the measurement of power consumption by the hardware and software in laptops with Windows operating system while the inbuilt battery status checker has been used to measure the battery consumption in an Apple ipad Air2 with IOS operating system. Some of the results in the experiments (particularly Section 4.1) conducted confirm the findings in existing research. However, further experiments are necessary to verify the findings in Section 4.2 and 4.3 by taking into consideration the experiment critique that have been discussed. This could be completed with additional use of other measure tools that have been discussed in Section 2, and also external measurement devices (e.g. multi-meter, etc…) which would yield more holistic experimental results. R